OBJECT ORIENTED PROGRAMMING IN THE JAVASCRIPT

Boris Bocharov, Maria Voevodina,
Nataliia Braterska, Anastasiia Dashkovska

Object Oriented Programming (OOP) means any kind of programming that uses a programming language with some object oriented constructs or programming in an environment where some object oriented principles are followed. At its heart, though, object oriented programming is a mindset which respects programming as a problem-solving dilemma on a grand scale which requires careful application of abstractions and subdividing problems into manageable pieces. Compared with procedural programming, a superficial examination of code written in both styles would reveal that object oriented code tends to be broken down into vast numbers of small pieces, with the hope that each piece will be trivially verifiable. OOP was one step towards the holy grail of software-re-usability, although no new term has gained widespread acceptance, which is why "OOP" is used to mean almost any modern programming distinct from systems programming, assembly programming, functional programming, or database programming. Modern programming would be better categorized as "multi-paradigm" programming, and that term is sometimes used [1-3]. This article is primarily aimed at modern, multi-paradigm programming, which has classic object oriented programming as its immediate predecessor and strongest influence.

Historically, "OOP" has been one of the most influential developments in computer programming, gaining widespread use in the mid 1980s. Originally heralded for its facility for managing complexity in ever-growing software systems, OOP quickly developed its own set of difficulties. Fortunately, the ever evolving programming landscape gave us "interface" programming, design patterns, generic programming, and other improvements paving the way for more contemporary Multi-Paradigm programming. While some people will debate endlessly about whether or not a certain language implements "Pure" OOP—and bless or denounce a language accordingly—this book is not intended as an academic treatise on object oriented programming or its theory.

Instead, we aim for something more pragmatic: we start with basic OO theory and then delve into a handful of real-world languages to examine how they support OO programming. Since we obviously cannot teach each language, the point is to illustrate the trade-offs inherent in different approaches to OOP.

JavaScript is a high-level, dynamic, untyped and interpreted programming language that is well suited for programming in object-oriented and functional styles. We will consider some specific features of programming in JavaScript on the example of the object "canvas"

Methods and Properties of the Canvas Element

The canvas element appeared in HTML5. You can get the canvas object using the getContext () method

```
<canvas id="canvas" width="300" height="100"></canvas>
<script type="text/javascript">
var canvas = document.getElementById('canvas'); // get an element
var cnv = canvas.getContext('2d'); // get a canvas object
</script>
```

Now we can work with the cnv object.

Pen and Brush Style

The pen is for outputting outlines, and the brush is for pouring.

cnv.strokeStyle - property defines the pen style;
cnv.fillStyle - property defines the style of the brush.
The value of these properties can be:

A stream in the style of hexadecimal color specifiers. For example, "# ff0000";

RGB function, type "rgb (red, green, blue)". For example, "rgb (255,0,0)" and "rgb (100%, 0%, 0%)";

RGBA function with alpha channel, "rgba (red, green, blue, alpha)". For example «rgba (255,0,0,0.5)»

An object created by one of the following methods:

createLinearGradient (x1, y1, x2, y2) - creates a linear gradient object;

createRadialGradient (x1, y1, r1, x2, y2, r2) - creates a radial gradient object;

createPattern (image, type) - fill pattern, type argument can take values:

repeat - repeat the pattern;

repeat-x - repeat the pattern with x;

repeat-y - repeat the pattern with y;

no-repeat - do not repeat the template.

```
cnv.strokeStyle = "#FF0000";
cnv.fillStyle = "rgba(255, 0, 0, 0.5)";
```

Also, except for transparency in the RGBA color value, you can define a global transparency value using the globalAlpha property. It takes a value from 0 to 1.0.

Contours

Methods for working with contours:

cnv.beginPath() - starts a complex contour;

cnv.closePath() - ends the complex path;

cnv.moveTo(x, y) - move the current pen coordinates to (x, y);

cnv.lineTo(x, y) - draw a line from the current coordinates to those specified in (x, y);

cnv.stroke() - traverse the contour with the stylus;

cnv.fill() - paint the outline with a brush;

cnv.quadraticCurveTo(cp1x, cp1y, x, y) - displays the second-order curve;

cnv.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y) - outputs the Bezier curve;

cnv.arc(x, y, radius, startAngle, endAngle, anticlockwise) - outputs the arc;

cnv.arcTo(x1, y1, x2, y2, radius) - adds an arc to the contour;

cnv.clip() - adds the region of the current path to the current region in which output is possible.

cnv.isPointInPath(x, y) - determines whether the point is inside the path, returns true or false.

Quadrilaterals

Methods for working with quadrangles:

cnv.rect(x, y, width, height) - displays a quadrilateral;
cnv.fillRect(x, y, width, height) - the filled quadrilateral;
cnv.strokeRect(x, y, width, height) - the contour of the quadrilateral;
cnv.clearRect(x, y, width, height) - cleaning the quad.

Under the quadrangles, it is understood here is not just a visible figure. For example, the clearRect method can be used as an eraser.

Line styles

Properties specifying style lines:

cnv.lineCap - line view.
- square - square;
- round - circle;
- butt is the butt.

cnv.lineWidth - width of the line;
cnv.lineJoin - specifies how to connect the lines.
- round - with rounding;
- bevel - with bevel;
- miter - with anti-aliasing, (default).

Text

Methods and properties for working with text:

cnv.fillText (text, x, y [, maxw]) - prints text to specified coordinates, with maximum width maxw;

cnv.strokeText (text, x, y [, maxw]) - prints the outline of the text to the specified coordinates, with the maximum width maxw;

cnv.measureText (text) - returns a TextMetrics object that determines the approximate width of the specified text. (TextMetrics.width).

cnv.font - css a string of font parameters (For example: italic 30px sans-serif);

cnv.textAlign - sets the alignment of the text ("start", "end", "left", "right", "center");

cnv.textBaseline - definition of the baseline ("top", "hanging", "middle", "alphabetic", "ideographic", "bottom");

Coordinate transformation

Methods for converting coordinates:

cnv.rotate (angle) - rotates clockwise by the specified angle in radians;
cnv.translate (x, y) - linear coordinate shift;
cnv.scale (x, y) - scaling;
cnv.transform (m11, m12, m21, m22, dx, dy) - multiplies the current transformation matrix with the specified matrix;
cnv.setTransform (m11, m12, m21, m22, dx, dy) - sets the new transformation matrix.

Images

To apply an image in the canvas element, you need to get it in the beginning. There are several ways to do this:

1. Use the DOM by means of getElementsByTagName or getElementById.
2. Create images in the script:
   var img = new Image ();
   img.src = "myimg.png";
3. Implement using data: url

var img = "data: image / gif; base64, ... image data ...";
Further images can be displayed on canvas:

cnv.drawImage (img, x, y) - displays images;
cnv.drawImage (img, x, y, width, height) - displays images with scaling in the specified sizes;
cnv.drawImage (img, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight) - displays the parts of the image with scaling in the specified sizes.

Raster Data

Methods for working with raster data:

cnv.createImageData (w, h) - creates raster data of size w * h. Returns the ImageData object;

cnv.getImageData (x, y, w, h) - gets the raster data of the canvas in the specified rectangle. Returns the ImageData object;

cnv.putImageData (ImageData idata, x, y) - displays raster data on the canvas to the specified coordinates;

cnv.putImageData (ImageData idata, x, y, sx, sy, sw, sh) - print a portion of the raster data to the canvas to the specified coordinates.

Preserving styles and transformation matrices

cnv.save () - save the state;

cnv.restore () - restore the state.

It is also useful to get a string of the format data: url, which can be transferred to the server and save the image to a file. For this, the cnv.toDataURL () method is used, it returns the string data: url encoded with base64. By default, the data contains information about the PNG image, but you can also present the image in a different format, for example cnv.toDataURL ("image / jpeg").

Practical Work. Object Oriented Programming

Based on the web page (see below), create the following object constructors:

1. polyline_figure (inheritor of PROTO_figure).
2. polyline_filled (inheritor of polyline_figure) is a closed polygon, colored inside.
3. circle_filled (inheritor of circle_figure) is a circle that is shaded inside.
4. arc_filled (inheritor arc_figure) - a segment of a circle, filled up inside.

Fill color is the constructor parameters.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
```

```
<style>
  .myclass {
    background-color: #CCFFFF;
    color: #CC0000;
    border: solid;
    border-color: #0000FF;
    width: 500;
  }
  canvas {
    border: 1px dashed black;
  }
</style>
<title> John Smith </title>
<script>
// Write into container
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}

// read and write data form
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
```

```
    return this.frm.elements[nm].value;
  }
  this.set = function(nm,vl){
    this.frm.elements[nm].value = vl;
  }
}

//Prototype for all figures
function PROTO_figure(o,c,ss,lw){
  this._origen = o;
  this._cont = c;
  this._strokeSt = ss;
  this._lineWi = lw;
  this.SelfDrow = function(){
    this._cont.lineWidth = this._lineWi;
    this._cont.strokeStyle = this._strokeSt;
  }
}

// LINE. Inherit from PROTO_figure
function line_figure(o,c,ss,lw,p1){
  this._line_end = p1;
  PROTO_figure.apply(this, arguments);
  var parentSelfDrow = this.SelfDrow;
  this.SelfDrow = function(){
    parentSelfDrow.call(this);
    this._cont.beginPath();
    this._cont.moveTo(this._origen[0],this._origen[1]);
    this._cont.lineTo(this._line_end[0],this._line_end[1]);
    this._cont.stroke();
  }
}
```

```
// TRIANGLE. Inherit from PROTO_figure
function triangle_figure(o,c,ss,lw,par){
  this._tr_end = par;
  PROTO_figure.apply(this, arguments);
  var parentSelfDrow = this.SelfDrow;
  this.SelfDrow = function(){
    parentSelfDrow.call(this);
    this._cont.beginPath();
    this._cont.moveTo(this._origen[0],this._origen[1]);
    this._cont.lineTo(this._tr_end[0],this._tr_end[1]);
    this._cont.lineTo(this._tr_end[2],this._tr_end[3]);
    this._cont.closePath();
    this._cont.stroke();
  }
}

// CIRCLE. Inherit from PROTO_figure
function circle_figure(o,c,ss,lw,r){
  this._cr_rad = r;
  this._st_angl = 0;
  this._end_angl = 2*Math.PI;
  PROTO_figure.apply(this, arguments);
  var parentSelfDrow = this.SelfDrow;
  this.SelfDrow = function(){
    parentSelfDrow.call(this);
    this._cont.beginPath();

this._cont.arc(this._origen[0],this._origen[1],this._cr_rad,this._s
t_angl,this._end_angl);
    this._cont.stroke();
  }
```

```
}
// ARC. Inherit from CIRCLE_figure
function arc_figure(o,c,ss,lw,r,an1,an2){
  circle_figure.apply(this, arguments);
  this._st_angl = an1;
  this._end_angl = an2;
}
</script>
</head>
<body>
<script>
// functions
function process(){
  var scmd = myf.get("gr_obj");
  scmd = "new "+scmd;
  cnt.add(scmd+"<br>");
  var fig = eval(scmd);
  fig.SelfDrow();
}
/*
line_figure([200,200],context,'#ADFF2F',20,[400,400])
triangle_figure([10,10],context,'#8B008B',5,[400,400,5,490])
circle_figure([250,250],context,'#FF0000',10,100)
arc_figure([250,250],context,'#FF0000',10,100,0,2)
*/
</script>
<form name="myForm">
Graphic object: <input name="gr_obj" type="text" size="80"
value="line_figure([200,200],context,'#ADFF2F',20,[400,400])"><br
/>
<input name="Process" type="button" onClick="process()"
value="Process">
```

```
</form>
<hr />
Report:
<div class="myclass" id="myDiv">result</div>
<br />
Graphic area:
<hr />
<canvas id="myCanvas" width="500" height="500"></canvas>
<script>
// Globals
var elem = document.getElementById("myDiv");
var cnt = new HTML_Container(elem);
var myf = new FORM_data("myForm");
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");
cnt.clr();
</script>
</body>
</html>
```

References:

1. Бочаров Б.П. Інформаційні технології в освіті : монографія / Б.П. Бочаров, М.Ю. Воєводіна; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків: ХНУМГ ім. О. М. Бекетова, 2015. – 197 с.

2. Bocharov Boris. AUTOMATIZED WEB PAGES PARSING AND CREATION / Boris Bocharov, Maria Voevodina // Information technologies in education: electronic supplement to the journal "Educational Institutions Libraries". – 2017. – N5, p. 1-5.

3. Bocharov Boris. BASIC CONCEPTS OF THE JAVASCRIPT / Boris Bocharov, Maria Voevodina, Nataliia Braterska, Anastasiia Dashkovska // Information technologies in education: electronic supplement to the journal "Educational Institutions Libraries". – 2018. – N7, p. 1-48.